U.S. DEPARTMENT OF COMMERCE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
NATIONAL WEATHER SERVICE
NATIONAL METEOROLOGICAL CENTER

OFFICE NOTE 154

AFOS Graphics Created by NMC

Joel Nathan
Automation Division

JUNE 1977

Section I - Formation of Scrambled AFOS File

The purpose of this Office Note is to document in detail NMC's procedure which produces AFOS graphics in vectorized code from the existing scan line format which produces the automated FAX/VARIAN maps. The first and most important aspect of this procedure is the "scrambled AFOS file", which the CNTR package produces. This file contains pieces of contours in delta-X, delta-Y format that are output onto a file as the CNTR program scans through a line in the map. The vectors have a label attached to them so that they may be identified as belonging to a certain contour. The starting position of these contours are output along with their value. The processing code pieces the vectors together to form AFOS contours.

The scrambled AFOS file is output using direct access I/O with a record size of 128 (4-byte) words. The last word of each record contains the record number and the first word contains the number of the next record in the sequence. The very first record is an identification record. (Please keep in mind that each FAX/VARIAN map may consist of as many as three meterological fields. Each of these fields represents a complete map when being displayed on AFOS, so each FAX/VARIAN map will represent one, two or even three AFOS maps.) Word six of the I. D. record will contain the record number of the first record in the sequence which contains the actual vector data for field one. If there are two or three fields, their respective starting locations will be contained in words seven and eight. The next available words will contain unique subset numbers for each field. These numbers will be matched with the proper labels contained in the alpha-numeric file.

For each field there is a set of three identification records. The first of these records contains the starting position of each contour. The next record contains a flag that points to the position in the vector file where the vector search for each contour shall begin. And the third record contains the actual value for each contour. The very last record in this set will contain a number that will point to a record that will either contain an end of maps message or will start another identification chain if there are more maps to be processed.

Since there are only 128 words in the record, we are limited to processing 128 contours. This limit is sufficient for most maps; however there are instances where we need to be able to process more contours. This is accomplished in the following manner. Looking back to the very first identification record there may be flags in the words immediately following the subset numbers which point to supplemental identification records, allowing us to have an additional 128 contours. These supplemental records are organized just as the others, and the vectors that they refer to are merged with the vector file. This completes the explanation of the scrambled AFOS file, but for a more detailed description of the vector file refer to Appendix I.

Section II - Processing of Scrambled AFOS File

This section describes the process of unscrambling the contour file explained in section one. It further explains how label information is merged in with the contours and output in an acceptable AFOS format. The first thing we need to do is read in all the records in the scrambled AFOS file for one AFOS map. Then we read in all the alpha-numeric data for the same map. This data contains identification information, contour labels, and titles. The subset number is contained in word two of the alpa-numeric file (see Appendix two for a description of this file.) We check to see if this number matches the subset number in the scrambled AFOS file. If they match, we check our master file of subset numbers for this entry and match it with its proper AFOS PIL number. We save this number along with the date (located in words five and six) because they must be put into the header for transmission. Each FAX/VARIAN map has a particular background associated with it. For FAX/VARIAN these backgrounds are generated with different orientations. For instance, the standard LFM background is generated with verticle longitude of 105° W, and the standard PE background is 80° W. However, in the current AFOS system we have a very limited number of backgrounds in existence, and they are all generated with verticle longitude of 100° W. Therefore, we must rotate the data to fit the background. The data also must be scale and offset to the background. These constants are determined by the particular background (contained in words 16 and 17) used for FAX/VARIAN. Now we actually start forming the alpha-numeric information by first issuing an instruction to get into character mode. The file contains EBCDIC data (labels and titles) along with its I, J coordinates. (see writeup of Appendix II for detailed description of the alpha-numeric file.) This data is transformed into ASCII text to meet AFOS requirements. The starting position of a string of data across the display is rotated, scaled, and offset with the constants referred to above. Succeeding characters in the string are displayed horizontally from the starting position across the screen. When a new string is found we output a new starting location followed by the character information, each character representing one byte of data. (See writeup of Universal Graphics Format for details of coding requirements.)

Now that the character information is done we must start forming the contours. We issue an instruction to get us into relative vector mode (see writeup of UGF.) There are several different coding techniques that are available; however, relative vectors is currently the only method that we are using for operational AFOS products. (See Appendix three for a more detailed discussion of all the formats). For each contour we have an initial position I, J which we rotate, scale and offset as above. We then go through the scrambled AFOS file and piece the vectors together. The logic is similar to that mentioned in Appendix one. Each vector that is output is a relative vector, i.e. it is a displacement value from the previous point. The relative vectors are one (two-byte) word in length with six bits of actual data plus a sign bit for each delta. The high order bit for each delta is currently not used. All the relative vectors for one contour are placed in an intermediate array and are passed to a compression subroutine along with the vector count and the initial position. This routine examines in groups of three all the points along the

contour formed by the vectors. It then tests for the curvature of the vector to see if the middle point along the contour can be eliminated. Then the two shorter vectors connecting three points can be replaced with one vector using two points. However, since the length of our vectors is only six bits, the longest vector we may have is 63 pixels. After the entire contour is compressed we return with the new vector count. We repeat this procedure for each contour until they are all processed.

We are now left with a message of alpha-numeric data followed by contour data. There are certain AFOS transmission formats that require us to block the data in a particular way. Each AFOS block consists of 256 bytes of actual data plus header and trailer information. (See Appendix four for details of communications.) The message is blocked into the proper format with header and trailer information and returned with the new byte count. NMC's interface operation, which transfers the message from the 360/195 to the 360/40, requires the data to be output in blocks that are 1280 bytes long. Therefore, five AFOS blocks will fit exactly into one communications block. The processing program is thus required to output the completed message in 1280-byte blocks to some disk area on the 360/195. Then the program begins another map and starts the process over again. Meanwhile the AFOS message gets called up through the interface to the 360/40, and from there is passed through the interdata and then to the NOVA at Suitland. Finally the message is transmitted to the NOVA at Gramax and stored at the AFOS facility. Now the map can be displayed by simply punching in the proper PIL number. The background is already stored at the facility and will automatically be identified when the appropriate PIL number is called up. To display a FAX/VARIAN map that has two or more fields, we display one field by calling up its PIL number and then use the overlay feature to call up the other fields. Hopefully the preceeding discussion will be of assistance to any user that wants to produce AFOS maps from an existing system that is in scan line format.

Appendix I - Formation of Vector File

The vector file is divided into 128-word records that are linked together with identification numbers as described in section one. Each word in the vector file is divided from left to right in the following manner:

Byte one is the command byte. The command byte identifies this word as a starting point, a finishing point, or an intermediate point. If it is a starting point, it also determines whether it is a forward search or a backwards search. And it could also tell us in the middle of a search that we are to reverse directions. This may sound complicated now, but it will become clearer as I explain how it works with the other parts of the word.

Byte two contains the line label, and it identifies each word as belonging to a particular contour.

Bytes three and four contain respectively the delta-x, delta-y relative vectors.

It is probably easiest to explain how the file works by taking a contour and following the logic necessary to piece it together. In the second identification record we have the flags that tell us where to begin our vector search. In words one through 128 we have a value, X, that tells us to look at the xth word in the vector file. In the second byte of this word we will find the line label which has permissable integer values of one through 127. This line label value corresponds to the particular word in the identification record that contains the value X. In other words suppose word six of that identification record contains the value four. Then the second byte of word four in the vector file has as its value five, and five is defined as the line label for this particular contour.

Now that we have identified the contour, we must examine the command byte (byte one) in order to determine how to begin our search. Suppose the command byte is four or twelve. This tells us that we have a forward vector search, and we scan forward through the file, starting at word five, looking for words with a line label of five, when we find a word with the correct line label, we again examine the command byte. If the command byte is 128, then we have found a word that has a vector value and is a piece of the contour. Bytes three and four contain the respective displacement values in the x and y direction from the previous position. Then we continue our search as before.

If we encounter a command byte value of two, then we must reverse our search. The value contained in byte four of this word is the new line label that we will be looking for as we reverse our search.

Looking back to when we started, suppose our original command byte was ten. In this case we would have a backwards search through the vector file. The search continues and will only be reversed if we find a command byte of four. If we are in a forward search, we will stop when we find a command byte of ten. For a backwards search the value is twelve.

In the case of a closed contour we would change direction several times and eventually we would point back to the same line label with which we began. Closed contours will begin with the command byte value of four, which takes us on a forward search. This contour will end with a command byte of four also, as long as the value of the new line is the same as the original.

The above description of the AFOS vector file, coupled with the discussion in section one, will give the user a good grasp of how AFOS contours are derived from FAX/VARIAN scan line format.

Appendix II - Formation of Alpha-Numeric File

Each AFOS map has a set of alpha-numeric data associated with it. Each map will have at least two records of data. The first one is a 50 word identification record that is flagged with the integer value of -1, in the first word. (See my memo of April 21, 1977 for a detailed description of this I. D. record.)

The succeeding records are 2048 words long and consist of actual label information in EBCDIC that is to be plotted on the map. As many records as are needed to supply all the label information are output. A record with word one equal to -7 signals the end of data for a particular map. The label records are actually divided into groups of two with the first word containing the I, J position, and the second word containing the label information. The actual two-word format is as follows:

1st Word

J (in scan lines)	(15 bits)
Temporary character-change function	(1 bit)
Priority	(3 bits)
I (in dots)	(13 bits)

2nd Word

Index to character set	(8 bits)
EBCDIC text	(24 bits)

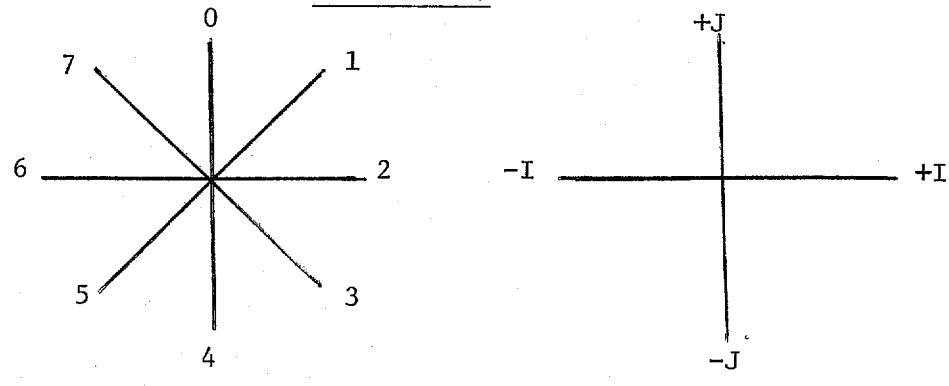
The temporary character-change function and the priority are not used for AFOS. Also the index to the character set is ignored currently, but with the addition of more AFOS character sets it will be utilized.

Appendix III - Other AFOS Graphics Formats

There are several methods of encoding AFOS graphics available to us. These methods enable us to strive for maximum data compression. Besides the relative vector mode that we are currently using, there is also the absolute vector. This is simply a line between points. Given any set of two points a line can be drawn between them, and this can be repeated up to $2^{16}-1$ times. This is not a very practical method for contours that are constantly changing direction, but for long, straight lines, such as the latitude and longitude lines on a background, this is an excellent method for maximum data compression.

Another method is the CPC (compressed pen command) string which assembles three-bit vectors into long strings to represent contour lines. The technique for drawing these lines is based on a fixed length vector which has eight possible directions of motion from any point.

CPC VECTORS



We have not done much experimentation with this method directly, however it is used as the first step in the process of forming the VEV (variable exception vector.) In this method a line is drawn based on the restriction that the maximum change in direction is +45°. There is an excellent discussion of how this can be done in the writeup of the "Variable Compact Vector Module for the NEDS-1 Display System" prepared for Fleet Numerical Weather Central. Once the CPC string has been smoothed it is possible to transfer it into a VEV string, which has a variable length of .01, .02, .03, or .04 inches, which is fixed for all of the vectors in the string. Along with the vector length we are given an initial direction vector which is the same as the CPC vector. We find this by adding the first three smoothed CPC vectors (see section 2.2.2.3 of NEDS writeup.) Once the IDV has been determined, the smoothed CPC string is processed in increments of up to 256 vector steps to compute the VEV string. The smoothed CPC string insures that the maximum deviation of adjacent vectors is +45°, that is a vector can turn either up or down 45 degrees from the format direction of motion or continue in the same direction. We can reduce the number of bits needed to describe the vector from three to one. This is accomplished by keeping a bit count (add or even) of the string, as well as the position and differences between the CPC and VEV strings. Introducing odd and even vectors allows us to do the following:

Even-numbered vectors are allowed to either continue in the same direction as the previous vector or change direction counterclockwise 45 degrees (+45°). Odd-numbered vectors are allowed to either continue in the same direction or change direction clockwise 45 degrees (-45°). A vector which continues in the same direction is called a trend vector and assigned a value of 0, and one that changes +45° is an exception vector, and is assigned a value of 1. By keeping track of the vector count and the CPC and the direction of motion we can make a decision whether to use a trend vector or exception vector to best follow the path of the CPC string.

It is very easy for the VEV to get off the CPC track, since it can't always turn at the proper time. Additionally, it is difficult for the VEV to make sharp turns. Minor deviations are expected, but the maximum difference allowed is four vector steps. When this happens this string is closed off and a long vector is drawn from the last position to the correct position. At one time we were encoding our products with the expectation that VEV would be the only acceptable format in the final AFOS system. Our experience shows that using a vector length of .01 inch we could not achieve nearly the data compression that we did with the relative vector with a simple curvature test for compression. Furthermore, the relative vector produced a smoother and more accurate product. However, the VEV is fully utilized only when used as a variable length vector, and it is entirely possible that the results would be much better if we varied the vector length.

In our current AFOS products we are using only relative vectors. This is because we found it to be the easiest and fastest way to convert from our current system. However, we expect that in the future we will incorporate several of these methods to afford us the maximum in data compression with the minimum of product distortion.

APPENDIX IV - NMC-AFOS Communications Block Formats

The following is the communications format for AFOS graphics data (All values are in hexadecimal):

Header For 1st Block	Byte #	1	2	3 4	5 6	7	8 9 10	11 12 13	14 15 16	17 18 19 20	21	22
	Symbol	SOH	XSN	MID	MAD	TPM	C C C	N N N	X X X	Mo Da Hr Mn	N1	LRC
	Definition	Start of header	Tx seq #	Message I. D. #	Message address #	Type, Prty, Mode	Station identifier	Product category	PIL number	Date in month, Day, Hour, Minute	# lines	red. chk.
	Value	01	00	40 00	00 00	00	4E 40 43	47 50 48	Unique value	Binary Value	00	00

Header for Blocks 2 to N-1	Byte #	1	2	3 4
	Symbol	SOH	XSN	MID
	Value	01	00	00 00

Header for Last Block	Byte #	1	2	3 4
	Symbol	SOH	XSN	MID
	Value	01	00	80 00

Trailer for Blocks 1 to N-1	Byte #	255	256
	Symbol	ETB	BCC
	Definition	End of Block	Block Check
	Value	17	00

Trailer for Last Block	Byte #	P-2	P-1	P
	Symbol	DLE	ETX	BCC
	Definition	Filler (If necessary)	End of message	Block check
	Value	10	83	00

P represents the last byte of the message.

Since the message must have an even number of bytes, the filler character is necessary.

The following characters have special meaning and can not appear as actual data:

DLE (10)
ETX (83)

Therefore, if a byte of data is encountered with either of the above values, it is replaced as follows:

DLE is replaced by DLE DLE (1010)
ETX is replaced by DLE FF (1000).